МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий

Кафедра параллельных вычислений

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«ВЫСОКОУРОВНЕВАЯ РАБОТА С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ»

студента 2 курса, 23202 группы

Пятанова Михаила Юрьевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
Кандидат технических наук
Владислав Александрович
Перепёлкин

Новосибирск 2024

СОДЕРЖАНИЕ

ЦЕЛЬ.....3
ОПИСАНИЕ РАБОТЫ 4
ЗАКЛЮЧЕНИЕ.....5
Приложения 1, 2, 3. 6

ЦЕЛЬ

1. Ознакомиться с программированием периферийных устройств на примере ввода данных с Web-камеры с использованием библиотеки OpenCV.

ОПИСАНИЕ РАБОТЫ

Для сборки проекта с библиотекой OpenCV было установлено средство автоматизации сборки программного обеспечения CMake. Была написана тривиальная программа вывода изображения с вебкамеры на экран (как исходный код, так и полученный вывод представлены в приложении 1).

Преобразованием изображения было выбрано аффинное преобразование (приложение 2), которые можно выполнить, используя функции:

getAffineTransform(const Point2f src[],const Point2f dst[])

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \texttt{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Находит матрицу аффинного преобразования, где

$$dst(i) = (x_i', y_i'), src(i) = (x_i, y_i)$$

src-координаты вершин треугольника на исходном изображении.
dst- координаты соответствующих вершин треугольника на целевом изображении.
Возвращает Mat map matrix – матрицу аффинного пребразования(оператор).

warpAffine(InputArray, OutputArray, OpMat, OutputArraySize)

Применяет матрицу аффинного преобразования OpMat на InputArray, записывая в OutputArray.
Ничего не возвращает.

Далее были проведены измерения количество кадров, обрабатываемое программой в секунду. Дана оценка доли времени, затрачиваемого процессором на обработку (ввод, преобразование, показ) видеоданных, получаемых с камеры (приложение 3). Так как показатели практически не меняются, выбираем данные произвольно:

Процесс	Доля (в %)
Ввод	21,9
Преобразование	53,3
Показ	3,8
Ожидание	21

При этом видео воспроизводилось в 19 кадров в секунду.

ЗАКЛЮЧЕНИЕ

Было проведено знакомство с программированием периферийных устройств на примере ввода данных с Web-камеры с использованием библиотеки OpenCV. Была измерена доля времени, затрачиваемого процессором на обработку (ввод, преобразование, показ) видеоданных, получаемых с камеры. Оказалось, что больше всего времени тратится на преобразование и ввод, что и логично, ведь аффинное преобразование подразумевает умножение большого количества матриц, а ввод, в основном, опирается на технические характеристики камеры.

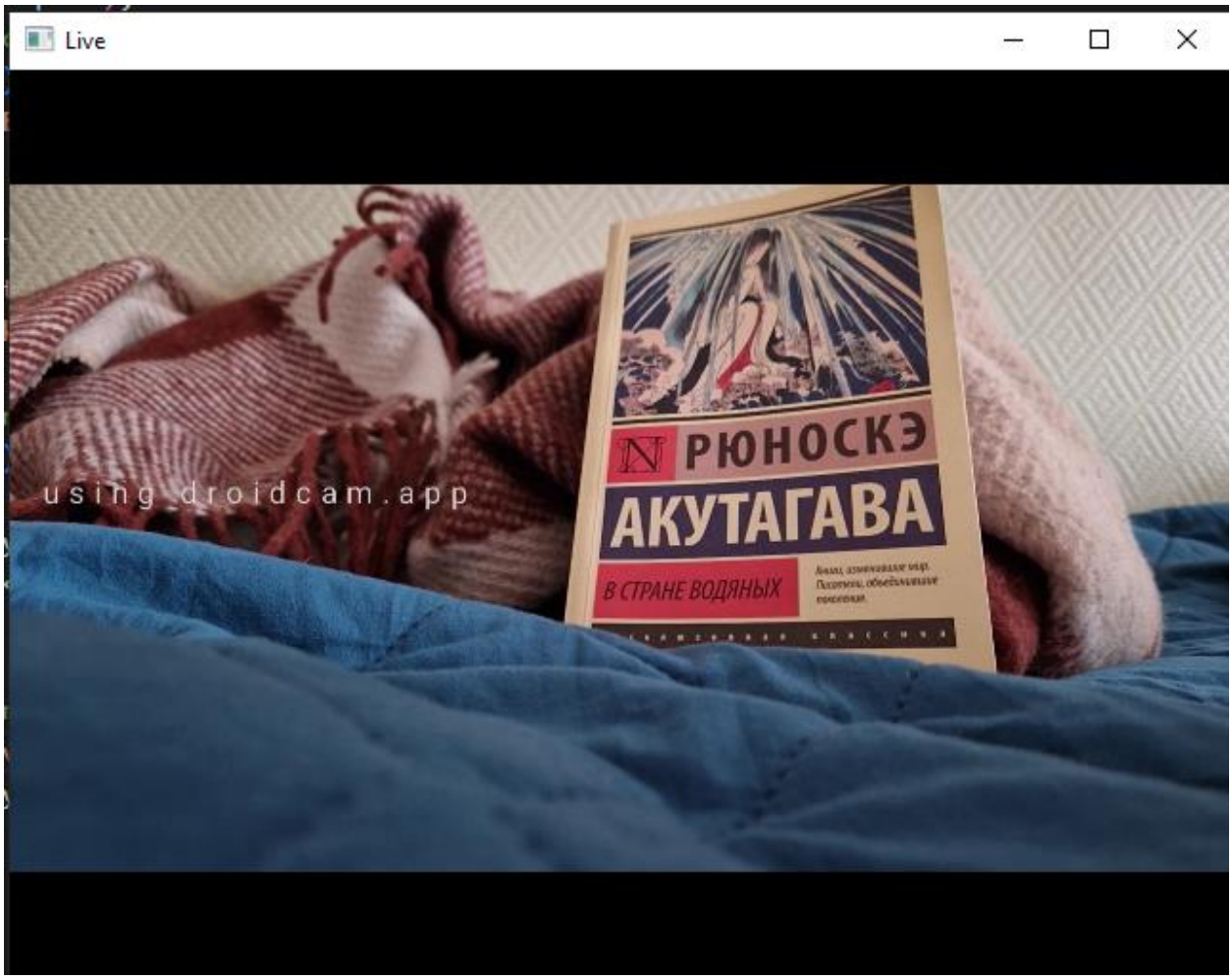
Приложение 1.

Команды для компиляции: cmake --build .\build\

```
cmake_minimum_required(VERSION 3.10)
project(lab_src)

set(OpenCV_DIR C:/Users/Pyata/Desktop/evm/labs/code/opencv/build)
find_package(OpenCV REQUIRED)

INCLUDE_DIRECTORIES(${OpenCV_INCLUDE_DIRS})

add_executable(${PROJECT_NAME} main.cpp)

target_link_libraries(${PROJECT_NAME} ${OpenCV_LIBS})
```

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>

int main() {
    cv::Mat frame;
    //--- INITIALIZE VIDEOCAPTURE
    cv::VideoCapture cap;
    // open the default camera using default API
    // cap.open(0);
    // OR advance usage: select any API backend
    int deviceID = 0;             // 0 = open default camera
    int apiID = cv::CAP_ANY;      // 0 = autodetect default API
    // open selected camera using selected API
    cap.open(deviceID, apiID);
    // check if we succeeded
    if (!cap.isOpened()) {
        std::cerr << "ERROR! Unable to open camera\n";
        return -1;
    }
    //--- GRAB AND WRITE LOOP
    std::cout << "Start grabbing" << std::endl
        << "Press any key to terminate" << std::endl;
    for (;;) {
        // wait for a new frame from camera and store it into 'frame'
        cap.read(frame);
        // check if we succeeded
        if (frame.empty()) {
```

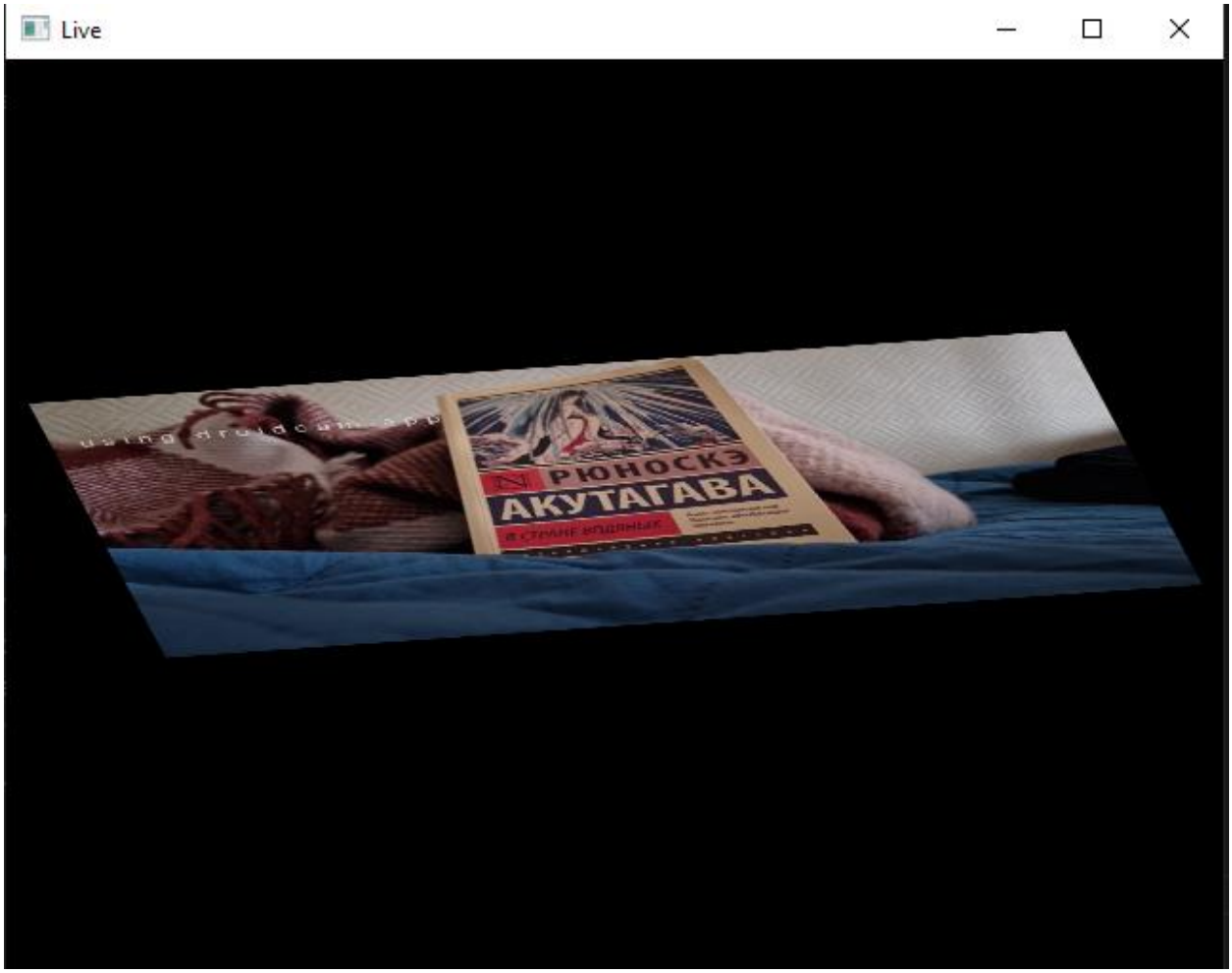
```
            std::cerr << "ERROR! blank frame grabbed\n";
            break;
        }
        // show live and wait for a key with timeout long enough to
show images
        cv::imshow("Live", frame);
        if (cv::waitKey(5) >= 0)
            break;
    }
    // the camera will be deinitialized automatically in VideoCapture
destructor
    return 0;
}
```



Приложение 2.

```
#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"

int main() {
    cv::Mat frame;
    cv::VideoCapture cap;
    int deviceID = 0;
    int apiID = cv::CAP_ANY;
    cap.open(deviceID, apiID);
    if (!cap.isOpened()) {
        std::cerr << "ERROR! Unable to open camera\n";
        return -1;
    }
    std::cout << "Start grabbing" << std::endl << "Press any key to
terminate" << std::endl;
    for (;;) {
        cap.read(frame);
        if (frame.empty()) {
            std::cerr << "ERROR! blank frame grabbed\n";
            break;
        }
        using namespace cv;
        Point2f srcTri[3];
        srcTri[0] = Point2f( 0.f, 0.f );
        srcTri[1] = Point2f( frame.cols - 1.f, 0.f );
        srcTri[2] = Point2f( 0.f, frame.rows - 1.f );
        Point2f dstTri[3];
        dstTri[0] = Point2f( 0.f, frame.rows*0.33f );
        dstTri[1] = Point2f( frame.cols*0.85f, frame.rows*0.25f );
        dstTri[2] = Point2f( frame.cols*0.15f, frame.rows*0.7f );
        // See detailed discription of the function in the lab work
discription.
        Mat warp_mat = getAffineTransform(srcTri, dstTri);
        Mat warp_dst = Mat::zeros(frame.rows, frame.cols,
frame.type());
        // See detailed discription of the function in the lab work
discription.
        warpAffine(frame, warp_dst, warp_mat, warp_dst.size());
```

```
        cv::imshow("Live", warp_dst);
        if (cv::waitKey(5) >= 0)
            break;
    }
    return 0;
}
```



Приложение 3.

```
#include <iostream>
#include <chrono>
#include <opencv2/opencv.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/highgui.hpp>
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"

int main() {
    cv::Mat frame;
    cv::VideoCapture cap;
    int deviceID = 0;
    int apiID = cv::CAP_ANY;
    cap.open(deviceID, apiID);
    if (!cap.isOpened()) {
        std::cerr << "ERROR! Unable to open camera\n";
        return -1;
    }
    using namespace std::chrono_literals;
    std::chrono::duration<long double> totalTime = 0s;
    std::chrono::duration<long double> inputTime = 0s;
    std::chrono::duration<long double> processingTime = 0s;
    std::chrono::duration<long double> outputTime = 0s;
    std::chrono::duration<long double> waitingTime = 0s;
    long long framesNum = 0;
    for (;;) {
        framesNum++;
        auto start = std::chrono::high_resolution_clock::now();
        cap.read(frame);
        auto end = std::chrono::high_resolution_clock::now();
        inputTime += end - start;
        totalTime += end - start;
        if (frame.empty()) {
            break;
        }
        using namespace cv;
        start = std::chrono::high_resolution_clock::now();
        Point2f srcTri[3];
        srcTri[0] = Point2f( 0.f, 0.f );
        srcTri[1] = Point2f( frame.cols - 1.f, 0.f );
        srcTri[2] = Point2f( 0.f, frame.rows - 1.f );
        Point2f dstTri[3];
```

```
        dstTri[0] = Point2f( 0.f, frame.rows*0.33f );
        dstTri[1] = Point2f( frame.cols*0.85f, frame.rows*0.35f );
        dstTri[2] = Point2f( frame.cols*0.15f, frame.rows*0.7f );
        Mat warp_mat = getAffineTransform(srcTri, dstTri);
        Mat warp_dst = Mat::zeros(frame.rows, frame.cols,
frame.type());
        warpAffine(frame, warp_dst, warp_mat, warp_dst.size());
        end = std::chrono::high_resolution_clock::now();
        processingTime += end - start;
        totalTime += end - start;
        start = std::chrono::high_resolution_clock::now();
        cv::imshow("Live", warp_dst);
        end = std::chrono::high_resolution_clock::now();
        outputTime += end - start;
        totalTime += end - start;
        start = std::chrono::high_resolution_clock::now();
        if (cv::waitKey(5) >= 0)
            break;
        end = std::chrono::high_resolution_clock::now();
        waitingTime += end - start;
        totalTime += end - start;
        if (totalTime >= 5s){
            break;
        }
    }
    std::cout <<"FPS: "<< framesNum / 5 << '\n';
    std::cout <<"Input time: "<< inputTime.count() / framesNum <<
'\n';
    std::cout <<"Processing time: "<< processingTime.count() /
framesNum << '\n';
    std::cout <<"Output time: "<< outputTime.count() / framesNum <<
'\n';
    std::cout <<"Waiting time: "<< waitingTime.count() / framesNum <<
'\n';
    return 0;
}
```

```
FPS: 19
Input time: 0.0115778
Processing time: 0.028297
Output time: 0.00196729
Waiting time: 0.0110399
```

```
FPS: 19
Input time: 0.0118222
Processing time: 0.0280914
Output time: 0.00198343
Waiting time: 0.0112757
```

```
FPS: 18
Input time: 0.0122334
Processing time: 0.0274964
Output time: 0.0020629
Waiting time: 0.0114624
```

```
FPS: 18
Input time: 0.0122325
Processing time: 0.0277813
Output time: 0.00205895
Waiting time: 0.0113535
```